AFRL-IF-RS-TR-2004-18
Final Technical Report
January 2004

M3T: MORPHABLE MULTITHREADED MEMORY TILES

University of Illinois at Urbana-Champaign

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. L175

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-18 has been reviewed and is approved for publication.

APPROVED: /s/
CHRISTOPHER J. FLYNN
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JANUARY 2004	**FINAL** **Jun 01 – Aug 03**

4. TITLE AND SUBTITLE

M3T: MORPHABLE MULTITHREADED MEMORY TILES

5. FUNDING NUMBERS

C - F30602-01-C-0078
PE - 62712E
PR - L175
TA - TI
WU - LE

6. AUTHOR(S)

Josep Torrellas, Ben Abbott, Ted Bapty, Bob Bassett, Hubertus Franke, Jose Moreira, David Ngo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Illinois at Urbana-Champaign
1304 W. Springfield Ave.
Urbana IL 61801

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFTC
26 Electronic Parkway
Rome NY 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-18

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Christopher J. Flynn/IFTC/(315) 330-3249 Christopher.Flynn@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The Morphable Multithreaded Memory Tiles (M3T) system proves that a polymorphous computing system with both polymorphous hardware and a complete polymorphous software environment is orders of magnitude more cost-effective than conventional computing systems. The cornerstone of M3T is two main contributions: (1) novel architectural support that enables the processor cores and memories in an M3T chip to reconfigure at run time, allowing the processing architecture to morph or reconfigure itself into different "architectural templates" such as TaskScalar, VLIW (Very Long Instruction Word), MIMD (Multiple Instruction Stream and Multiple Data Stream), or Streaming engine, or even some combination of them as a program executes; and (2) a complete polymorphous software environment that includes synthesis tools for high-level design, a polymorphous low level compiler and code generator, and morphware fragments. TaskScalar is an architectural template where the processing components work in a tightly-coupled manner, executing very fine-grain tasks speculatively. Some of M3T's polymorphous features have been tech-transferred to the IBM Blue Gene/C (Cyclops) chip. To ensure commercial viability and defense interest, the primary demonstration application domain for the M3T system is speech processing.

14. SUBJECT TERMS

Polymorphic Computing Architecture
Morphware

15. NUMBER OF PAGES

41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

M3T: Morphable Multithreaded Memory Tiles

1. INTRODUCTION AND OVERVIEW

Malleable, polymorphous computing systems can provide solutions that are orders-of-magnitude more cost-effective than conventional ones through their ability to morph or reconfigure to match changing mission and scenario demands. Instead of adopting a hardware-first approach, these systems rely on post-silicon optimization in a way that both hardware and software cooperate in a constraint-sensitive environment.

The current state-of-the-art in malleable systems is largely represented by FPGA hardware. However, FPGAs are typically designed to enable the fine-grained programming of structures; they were not conceived to implement space-efficient, highly-optimized complex functional blocks such as floating-point units or memory hierarchies. Moreover, FPGAs tend to display low speed. Finally, FPGA systems lack the sophisticated polymorphous software support, in the form of morphable run-time systems, compilers, modeling, and synthesis tools, that would complement the hardware.

With the Morphable Multithreaded Memory Tiles (M3T) project, a novel approach to reconfigurable systems is explored. Adaptation at a coarse grain, using as reconfigurable blocks many general-purpose RISC cores interleaved with memory blocks in a chip is also explored. In addition, advanced polymorphous software support, where adaptivity is pervasively embedded into every layer is explored.

An M3T chip is composed of several general-purpose RISC cores and memory blocks placed in an interleaved manner (Figure 1). The hardware that interconnects the processor and memory modules and provides support for their communication and synchronization can be dynamically reconfigured in many ways through software as the program runs. As a result, the overall processing architecture morphs or reconfigures itself into different "architectural templates'' such as TaskScalar, VLIW (Very Long Instruction Word), MIMD (Multiple Instruction Stream and Multiple Data Stream), or Streaming engine, or even some combination of them sharing the

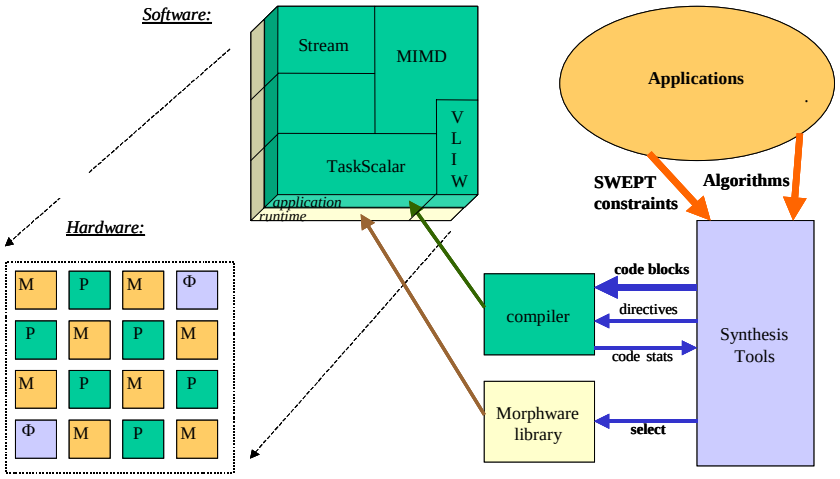


Figure 1: System with Morphable Multithreaded Memory Tiles (M3T).

chip as a program executes. TaskScalar is a novel template where the RISC cores work in a tightly-coupled manner, executing very fine-grain tasks speculatively. The TaskScalar template delivers very high performance for irregular codes (where neither superscalars nor chip-multiprocessors excel) and enhances programmer productivity.

An M3T chip has several reconfigurability axes including the number of processor and memory modules that participate in the reconfigurations and the characteristics of the processor modules (number, type, latency, and bandwidth of the functional units) and memory modules (levels of caching and organization).

The M3T architecture is supported by advanced polymorphous software that includes a set of modeling and synthesis tools, a compiler, and morphware. Based on the characteristics of the application to run, the modeling and synthesis tools generate a near-optimal (i) architecture and morphware configuration, and (ii) application partitioning and scheduling. The compiler generates code for M3T that can dynamically adapt to changing conditions. Finally, the morphware provides polymorphous, dynamic run-time support, including thread scheduling, memory management, and device drivers.

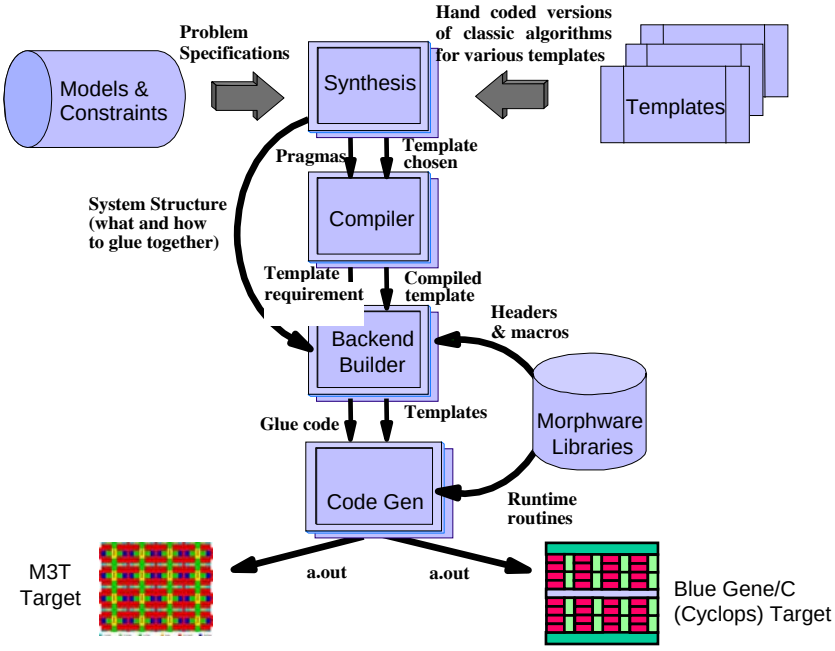


Figure 2: Comprehensive polymorphous software support in M3T.

To ensure technology transfer of our ideas, the M3T team includes leading research and development personnel from IBM and BAE SYSTEMS. The M3T work has been tightly-coupled with IBM's Blue Gene/C (Cyclops) chip design [CAS02]. This is evidenced by the fact that some of the polymorphous hardware and software features of M3T have been incorporated into the Cyclops chip.

In the following sections, we describe the proposed M3T system solution (Section 2), the methods, assumptions, and procedures followed (Section 3), background and alternative solutions (Section 4), the results obtained (Section 5), technology transfer (Section 6) and conclusions and recommendations (Section 7).

2. PROPOSED SOLUTION

In this section, the M3T's polymorphous micro-architecture is described (Section 2.1), along with its polymorphous software, which is composed of modeling and synthesis tools (Section 2.2), compiler (Section 2.3), and morphware (Section 2.4). Finally, an overview of the applications that were evaluated is provided (Section 2.5).

2.1 Polymorphous Micro-Architecture

The key novelty of M3T's micro-architecture is its ability to quickly morph the processing resources on the chip into a TaskScalar, VLIW, MIMD, or Streaming architectural template, or even a combination of them as the program executes. The TaskScalar is a novel template where the RISC cores work in a tightly-coupled manner, executing very fine-grain tasks speculatively. TaskScalar delivers very high performance in irregular codes such as SpecInt 2000 [SPE00], which have many accesses to memory through hard-to-analyze C pointers, loops with few iterations, many hard-to-predict branches, and data structures dominated by short arrays of records. In these codes, superscalars cannot find much instruction-level parallelism and chip-multiprocessors cannot extract more than one thread most of the time. In addition, TaskScalar enhances programmer productivity as will be shown in Section 5.1. The ability to morph is enabled by several special hardware support features provided in the architecture. Specifically, the M3T architecture goes beyond providing efficient support for the conventional *core-to-memory communication* and *core-to-core synchronization*. It also provides efficient support for *core-to-core data communication* and *core-to-core control communication.*

The M3T architecture supports core-to-core data communication by allowing a processor to read from or write to the registers of another processor. Core-to-core control communication is supported by allowing a processor to broadly control the data-path of another processor. These features allow the M3T architecture to morph into a TaskScalar, VLIW, or Streaming engine (Table 1).

Template	Core-to-Mem Communication	Core-to-Core Synchronization	Core-to-Core Data Communication	Core-to-Core Control Communication
TaskScalar	X	X	X	X
VLIW	X	X	X	X
MIMD	X	X		
Streaming	X	X	X	X

Table 1: Supports required for each architectural template.

The different architectural templates are implemented as follows:

MIMD
Each processor operates independently, executing its own stream of instructions. Communication occurs only through reading and writing words in memory. This template uses the fewest number of the polymorphic hardware features. It enables efficient execution of coarse-grain explicitly parallel code.

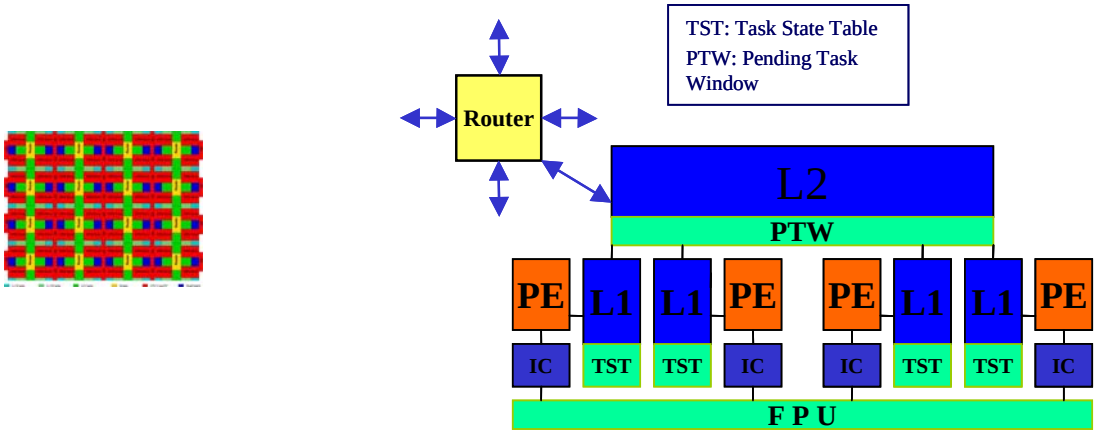
The MIMD execution model directly benefits from the underlying M3T multiprocessor architecture. Parallel applications already developed using a MIMD philosophy will run well on M3T. Furthermore, since all the cores are located on a single chip, communication between threads is inexpensive.

TaskScalar

The instruction stream is broken down by the compiler into very fine-grain tasks which are dynamically assigned to the processor cores. A task assigned to a core may not be eligible to execute because of a data or control dependence with a predecessor task that has not yet completed. When the dependence is resolved, the task is allowed to execute. This mode of execution requires the ability of a core to update the register structures of another (e.g. when passing the input arguments in a task spawn), control another core (e.g. stall that other core until the dependence is resolved), and quickly synchronize, if necessary. Overall, this template enables aggressive parallelization of irregular codes.

Parallel programs can utilize the MIMD template in M3T. However, the TaskScalar template is designed for situations when the application does not have enough coarse-grain parallelism, and using fine-grain tasks results in too much overhead. For these situations, the TaskScalar template provides fast synchronization and communication between cores. It uses two hardware modules called Pending Task Window (PTW) and Task State Table (TST). These modules, which are described next, enable a tightly-coupled architecture that efficiently executes very small tasks.

Cores collaborate to provide the necessary task-level parallelism. To enforce dependences between tasks, the TaskScalar template uses an architectural module developed for the M3T called the Pending Task Window (PTW). The PTW has an entry for each active task, which contains the address of the location that the task will update. The PTW checks for dependences between tasks and enforces them in a way similar to thread-level speculation. Specifically, every time that a reader task needs to read the address that a task will write, a message will be sent to the corresponding PTW. There, the hardware will enforce the following protocol. If the reader is a predecessor of the writer task, the reader will be allowed to proceed, reading the value that existed before the write. If, instead, the reader is a successor, then the read takes the value



(a): M3T chip (b): One of the 16 clusters in the 64-processor M3T chip

Figure 3: Organization of the PTW and TST hardware structures used in M3T.

produced by the writer or stalls if the data is not ready. This support synchronizes readers and writers.

Note that the PTW is not a centralized structure; it is distributed between cores. One PTW is associated with each cluster of processors in the M3T chip. For our base design, the M3T chip has 16 clusters of 4 processor elements each. Consequently, there are 16 PTWs per chip. The state of a task is stored in a Task State Tables (TST). It contains the registers and other state required to restart the task, like in thread-level speculation. Each processor has one TST. Figure 3 shows the organization of the PTW and TST hardware.

VLIW

Each core executes its own stream of instructions, but the cores operate in lockstep. The union of the independent streams is equivalent to a single VLIW stream. This template works best when the streams of the different cores are very similar to one another. Cores synchronize after every few instructions. During each step, at most one operation can be a branch. If the branch is predicted taken, the processor executing the branch uses core-to-core control communication to force all the other processors to jump to the new VLIW word. Furthermore, data may also be passed from the registers of one core to the registers of another. Finally, very fast synchronization support ensures that all processors execute in lockstep.

There are many scientific applications which have very little task level parallelism but exhibit large instruction level parallelism. If these applications were forced to run sequentially on a small core, the performance would be very poor. In large applications, this sequential code would limit the performance of the entire application significantly.

VLIW architectures can execute these codes well because loops can be pipelined and scheduled very tightly across all the functional units. In M3T, a small piece of code is distributed across multiple cores in order to utilize the hardware in the same way a VLIW architecture does. Each core computes part of the computation and synchronizes at set intervals with its neighboring cores. This results in a multithreaded lock-step execution of the program.

The VLIW template uses the M3T's synchronization support to synchronize in a handful of cycles. This provides a tightly coupled execution across cores. As a result, parallelization of loops and other structures with a large number of dependences is possible without suffering the cost of expensive synchronization operations.

Streaming

The M3T architecture also supports Streaming. The mapping of the Streaming Virtual Machine (SVM) [SVM03] to an M3T chip is as follows: (1) the Stream Execution Core is the set of CPUs; (2) the Kernel Program Memory is the instruction caches of the CPUs; (3) the Stream Register File is the shared multi-banked cache; (4) the DMA Engine is a plain CPU; and (5) the Control Thread is a plain CPU.

With this mapping, the workings of the SVM are as follows. The Stream Execution Core loads stream inputs from (and stores them to) the shared multi-banked cache, which acts as the Stream Register File (SRF). The Kernel Program Memory works trivially, as it is the instruction caches

of the processors in the M3T chip. The SRF allows read and write accesses from both the Stream Execution Core and the Direct Memory Access (DMA) engine, simultaneously. The DMA engine is just another CPU that generates addresses by strides, or indexed if the addresses are recorded in the SRF. It should be noted that the M3T's DMA engine can perform very complex operations, since it is implemented in the M3T architecture with a standard CPU. Finally, the Control Thread gives commands to the Stream Execution Core and the DMA Engine through the M3T's PTW. The Control Thread is notified of task completions (the VM_DONESYNC calls) in two ways: (1) it can build a tree of PTW requests, which allows pipelined streaming, or (2) through the use of barriers.

2.1.1 FFT As An Example

To illustrate the flexibility of supporting multiple architectural templates, an FFT will be used. An FFT can be graphically represented with a butterfly network as in Figure 4, where the nodes are operations and the edges data transfers. Each node in a butterfly computation involves reading two complex data operands and one complex coefficient, performing ten real operations, and storing two complex results.

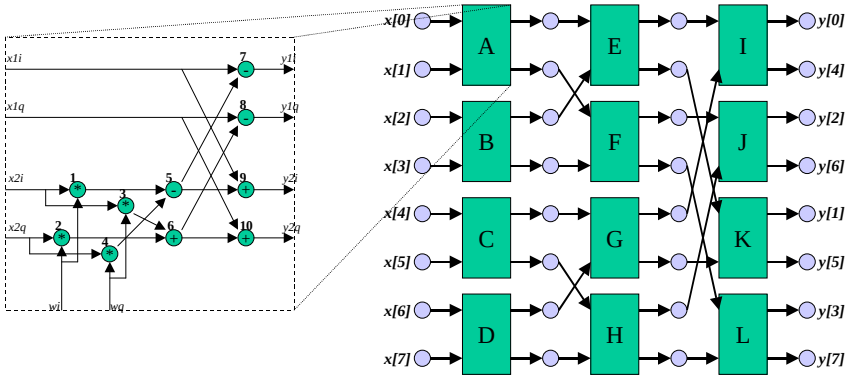


Figure 4: 8-point radix-2 FFT example.

Depending on the number of processors available, and the cost of computation and communication, different templates may be preferable. For example, if we have few processor modules, we can use the VLIW or TaskScalar template. In this case, the nodes represent ALU instructions. We distribute the instructions into as many groups as issue width we want, and use software pipelining to fill up slots. While either VLIW or TaskScalar would work, VLIW is preferred because all dependences are known statically.

If many processors are available and communication is cheap, the Streaming template can be used. In this case, a processor is assigned to each node in the butterfly network and the communication is forced to proceed synchronously. If, instead, fewer processors are available, one processor is assigned to each row in the butterfly network and the MIMD template is used. It is possible that much time is wasted synchronizing. However, the MIMD template may make sense if a very large FFT is needed. In this case, a large chunk of the network is assigned to each processor and synchronization is negligible.

2.2 Polymorphous Modeling and Synthesis Tools

To promote migration to the newest, evolving hardware, the M3T solution uses a software-first approach. This is in contrast to the current practice in building resource-constrained, high-performance systems, which involves implementing software that is hardwired to a specific *(soon-to-be legacy)* hardware architecture. The M3T approach requires a logical and physical separation of application specification from hardware implementation. It also requires a flexible application specification to capture a design space, expressing flexibility for future hardware capabilities. For M3T, we have generated tools that have the capability to *synthesize for heterogeneous architectures* and allow rapid *re-mapping to multiple architectures* during run time.

Figure 5 shows an example of the concepts captured in our modeling tools. Software component models are heavily annotated. The hardware modeling captures substantial details of components and their mapping into architectural templates.

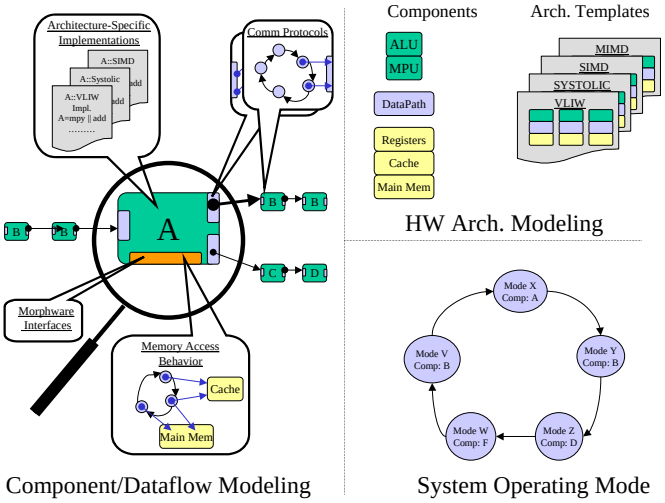


Figure 5: Concepts captured in the modeling tools.

For our polymorphous modeling and synthesis tools, there are two issues to consider: (i) design space representation and (ii) design space exploration and system synthesis. We consider each one in turn.

2.2.1 Design Space Representation

Our tools support multiple experts contributing to the system design. M3T uses a natural, solid set of formalisms and linkage mechanisms for polymorphous design capture. We describe them next, along with our methods for synthesizing systems directly from these models.

Asynchronous Dataflow is effective in capturing a wide range of embedded, performance-critical systems [EDW97]. A dataflow representation has three advantages:

1. Parallelism is retained. The algorithm is not artificially serialized. Since nodes in the graph are independent, except as explicitly defined by inter-node connections, nodes can be arbitrarily assigned to independent memory spaces (processors). The representation is architecture-independent.

2. Control is implicit. Process scheduling is determined by data availability (and node priority). In fixed-rate systems, processes can be scheduled statically, for maximum efficiency and guaranteed (by construction) timing.

3. Domain match. Dataflow concepts map to signal and speech processing nomenclature. This is evidenced by the popularity of Mathwork's Simulink.

Hierarchical finite state machines (FSM) are an accepted method for capturing dynamic model system behavior [HAR]. In this capacity, they can be used to represent the individual modes of the system under development, along with the mode-varying system requirements. Arbitrarily complex behavior can be specified and managed.

Hardware architectural models capture aggregate speed, memory, connectivity, and type of a processor. Prior research developed methods for capturing and managing macro-scale architectural adaptation. These models used an attributed, hierarchical block diagram, familiar to designers as schematics. As M3T's tools manage, allocate and configure processors to a much finer level (both temporal and physical), we require a significant depth of information to capture morphable processors.

System and application constraints are key to specifying implementation requirements and building a system that responds to those requirements. Constraints cross-cut the system design, tying system modes to algorithms implementations, implementations to architectures, and architectures to hardware components. Timing constraints establish temporal requirements of the system. Constraints offer a way to manage and explore entire design spaces, an impossible task with standard optimization techniques.

2.2.2 Design Space Exploration and System Synthesis

System synthesis converts these models, representing a large design space, into a set of feasible design implementations. The models capture a vast quantity of design solutions as a design space. Synthesis must find at least one that satisfies all of the system constraints. M3T's design tools have developed methods for navigating large design spaces, and finding viable solutions for reconfigurable systems. Our work represents a significant extension from past Adaptive Computing Systems (ACS) work, since the design space contains several new dimensions in micro-architectural flexibility.

Generating the implementations for the morphable architecture involves first selecting an architectural template. Computational components are mapped to the entities of the template. The code and hardware configuration instructions/tables are synthesized so the system produces

the proper computations at the proper time. The code contains system-level clues to enable the compiler to generate correct and efficient operations (Section 2.3).

Generating the time-critical communications infrastructure involves: selecting the physical communications medium; matching protocols for each individual pair of producers/consumers; and synthesizing a program/configuration for the communication hardware/software components. This relies on having an accurate, formal description of the communication protocol of a component (stored in the models) and a method for synthesizing near-optimal communications engines.

The morphware components are configured to produce a morphable kernel. This kernel is customized to application requirements, software dataflow information, and target hardware architecture. A specific instance of the morphware is driven by the application and the architecture, and can take several shapes during the functional lifespan of the system. The modeling and synthesis tools optimize the morphware kernel for the application.

Finally, embedded systems require **guarantees of correctness**. Formal verification methods are insufficient to prove correctness, so validation testing is employed. Testing is never fully satisfactory, due to time limitations. A continual verification process is needed. Built-in validation must not add significantly to the cost/size/power of the system, so minimal-load testing procedures must be inserted into the production version of a system. This minimal loading is achieved by testing only critical parameters, identified as system constraints in the design space. On-line checking algorithms are generated to verify system constraints. Since the checking infrastructure is created along with system synthesis, the satisfaction of real-time and other constraints takes the load of checking infrastructure into account.

2.3 Polymorphous Compiler

The M3T polymorphous compiler is based on the Stanford University Intermediate Format (SUIF) [HAL96] compiler. It is a source-to-source C compiler that works closely-tied with and subordinated to the modeling and synthesis tools just described. The compiler has two main functions, namely to generate M3T code and to support the tools. We consider each function next.

2.3.1 Generation of Code for M3T

The M3T compiler is responsible for generating code that will be executed on different architectural templates. Because automatic selection of templates is difficult, the compiler needs hints to decide on which template a particular kernel should execute. Rather than extend the language with special constructs, the compiler relies on pragmas, either inserted by the programmer or generated automatically by synthesis tools. Such pragmas will be used by the compiler to generate the correct code. The pragmas employed in the source code provide a variety of information to the compiler. There are three groups of pragmas.

Architecture reconfiguration pragmas. These are the most important. They indicate how the architecture should be reconfigured at run time. Reconfiguration can occur at the beginning of the application execution and also at different points during the application.

Code breakup pragmas. They indicate how the code should be broken down into modules to be scheduled as a unit. The code in a module should have a uniform behavior and good locality. Depending on what architecture template the module is expected to run on, the compiler optimizes its data structures and paths differently.

Module scheduling pragmas. They indicate how, when, and where the module should be scheduled. For example, these pragmas may indicate that a module be executed when a certain condition occurs on a certain architectural template. In addition, these pragmas may also specify the synchronization and communication between this module and other modules.

Typically, the compiler fully specifies the run-time conditions of the application. However, there are cases when there is not enough information at compile time to make all the decisions. In this case, pragmas indicate the several plausible choices and command the compiler to generate adaptive code. Specifically, the code is generated with several (computation alternative, architectural template) pairs. When the system reaches the code at run time, it measures the run-time conditions and, depending on them, selects one of the pairs.

In our implementation, we have organized the M3T compiler as a set of *transformers*, one for each architectural template. Figure 6 shows the structure of the M3T compiler.

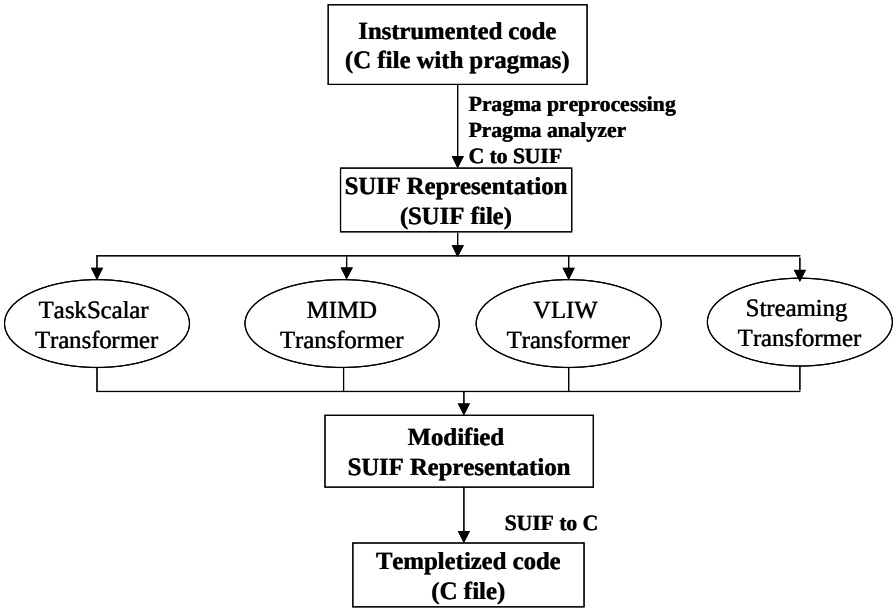


Figure 6: Structure of the M3T compiler.

2.3.2 Supporting the modeling and synthesis tools

The compiler has very detailed models of the different architectural template. These models include the latencies and bandwidths of functional units, wires, caches, and other components. In addition, the compiler can easily estimate how long a piece of code will take by compiling the code and generating machine instructions with all the compiler optimizations added. These two capabilities are exploited by the modeling and synthesis tools by including the compiler in the loop of hardware/software generation.

In this case, the tools call the compiler with several (computation alternative, architectural template) pairs. The compiler uses all the information that is available statically, to predict memory access patterns and resource contention. With all this information, it returns the best pair to the tools. Note that, when the compiler makes decisions, it does so fully aware of the morphware shapes that the code will find at run time (Section 2.4). In particular, it knows about the thread scheduling and memory management used for the particular architectural template it compiles for.

2.4 M3T Morphware

The M3T morphware provides the glue that marries a high-level application program to the dynamically changing low-level M3T architecture. The inner guts of the morphware are a set of modules of a real time variant of the popular Linux operating system, which provide a Posix-based Application Program Interface (API). The shape of the morphware at any one instance is driven by two forces. From the top, the applications require various subsets of the API points to be available at any one time. From the bottom, the M3T template is changing and concepts such as physical input/output, network interface, interrupt servicing, data-paths, and processor registers and state are a moving target.

The M3T morphware support is driven by a set of conflicting requirements.

1) The software first approach requires a consistent API for basic system services.
2) The asynchronous dataflow paradigm utilized by the modeling tools (Section 2.2) mandates a streams-like interface for glue between the computational nodes. Efficient implementation of the inter-computational node glue is highly dependent upon the processor architecture at any moment. Even the data inside these streams can change.
3) Architectural changes can occur dynamically, driven by events detected at the higher level, or they may change dynamically based on lower-level scheduling events or performance enhancement opportunities. Synchronization and coordination services between the lower and upper architectures are thereby required dynamically.
4) As the architecture changes, the performance requirements for the morphware remain. The M3T morphware must provide a run-time performance to meet many application domains, operating with minimal resources. The M3T morphware strives to be like a special hand-coded kernel for each instance of its higher and lower level interfaces.
5) A monolithic pre-built morphware image supporting all possible system scenarios, states, and shapes could be enormous.

The upper shell of the M3T morphware allows it to present services in a fashion typical of conventional operating systems. The core of the morphware kernel is a real-time scheduler. The M3T morphware includes system initialization, debugging support, exception handling (like signals), system calls such as time, inter-process communication, shared memory access, synchronization primitives, and memory management. As well, concepts of I/O are supported through device drivers. The device drivers themselves are able to deal with interrupts, DMAs, and programmed I/O regardless of the architecture shape. Table 2 describes some of the M3T morphware shapes.

	Scheduler	Memory Management & System Calls	Device Drivers
Task-Scalar	- Scheduler classic form - Process state: stack, task control block, memory spaces (per process) - Critical section implemented by disabling interrupts	- Single memory pool - Mutual exclusion via critical sections for memory access and system calls	- Utilize programmed I/O. - Interrupts used for external event synchronization - Architecture can utilize instruction unit like a DMA or I/O processor
MIMD	- Micro-scheduler for each instruction unit - Shared memory space guarded through multi-instruction unit mechanism for critical sections	- Single shared memory pool - Mutual exclusion based access across all instruction units for memory access and system calls	- Utilize programmed I/O - Architecture can utilize instruction unit like a DMA or I/O processor - Synchronization need for valid/fair device access - Interrupts must be dispatched to various instruction units
Streaming	- Instruction units self-scheduled by flow of data - Critical sections and scheduling only make sense if there is a guiding host execution unit - Context switch requires draining array and restructuring the rules	- With a hybrid approach (something more like a MIMD with lockstep communication), system calls and memory management follow the rules of the MIMD architecture.	- Device drivers managed based on the array - Interrupts are more like synchronization points causing data flow at the edges of the array
VLIW	- Like TaskScalar with much larger context	- One pool like TaskScalar	- Device drivers like TaskScalar - Interrupts at a single instruction unit must schedule the control lines to force all of the units to branch to the interrupt service

Table 2: Subset of M3T morphware shapes and issues.

2.5 Applications

While a polymorphous architecture such as M3T can be used for different application domains, we have focused on speech processing. This domain was chosen for its importance to both commercial and military communities. We have been especially keen to make the case for the potential commercial interest of M3T (and other polymorphous computer architectures) to IBM. From many conversations with IBM researchers and developers, we concluded early on in the project that speech processing makes the most commercial sense for a chip like M3T. The M3T and Cyclops chips will be more cost-effective solutions than current chips for speech processing, due to their density of compute power and their versatility.

Speech/voice processing applications need high compute capabilities and sizable (but not enormous) memory sizes. The Cyclops and M3T chips provide enough compute power density. For example, a Cyclops chip can provide 64000 MIPS, which is roughly enough to support 32 *speech* channels in real time. As for the memory requirements, we are counting on the caches to capture the first-level working sets and the 4 Gbyte off-chip memory to capture the memory footprint of the applications. Note that this application domain inherently contains a lot of parallelism, so we can tolerate the off-chip memory demands with parallelism.

The commercial application (Sphinx) [RAV96] and the defense application (Language Identifier or LID) were chosen as the primary speech processing applications that could demonstrate the polymorphic capabilities of M3T.

IBM researchers have also focused on network processing (TCP processing, cryptography, protocol conversion, etc). While they kept some of their evaluations and algorithms confidential, they have found that a chip like Cyclops is very relevant and potentially cost-effective for these types of workloads. They are in designing a Cyclops-light version of the chip for network processing.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

In this section, the methods, assumptions, and procedures that were used to evaluate M3T are described.

3.1 Polymorphous Micro-Architecture

To evaluate M3T's polymorphous architecture, a detailed software simulator of the M3T hardware, with all its architectural templates has been developed. This simulator is able to run program executables produced by the chain of M3T polymorphous software tools (Figure 2). With such support, we have been able to get a very accurate glimpse of how the actual M3T chip would perform.

In addition, for power consumption evaluation, the M3T simulator has been enhanced with a power simulator developed at the University of Illinois as part of the FlexRAM project [HUA00,KAN99]. With this support, the power consumption of the different modules of the M3T architecture can be estimated.

As shown in Figure 2, the M3T polymorphous software tools can also generate executables that run on the IBM Blue Gene/C (Cyclops) simulator. This capability has enabled the evaluation of our polymorphic concepts on the Cyclops design [CAS02]. Each Cyclops chip has 384 concurrent threads, 8 Mbytes of memory, and can achieve up to 24 Gflops. The latest version of the Cyclops architecture now includes several polymorphous features originally proposed for M3T, including: (i) very fast barrier synchronization (wired-OR), (ii) support for synchronization groups, (iii) fast access to other threads' local memory, and (iv) interest caching.

These four features are described in detail in [CAS02]. The very fast barrier synchronization allows all threads to synchronize in a barrier in a handful of cycles. This support is required for the TaskScalar, VLIW, and Streaming templates. It can work with synchronization groups, whereby groups of threads synchronize independently in different barriers. This support is required in M3T to space-share the chip between several different templates. The fast access to other threads' local memory is used to support the ability of processors in the TaskScalar template to write to other processors' Task State Tables and read from other processors' Pending Task Windows. It is also used to support the fast communication in the Streaming template between the Control Thread, Stream Execution Core, and DMA. Finally, the interest caching is a special support that enables a programmer to specify, when a processor is loading a stream of cache lines, where these lines are loaded. For example, these lines can go to a subset of processor caches (an "interest group''), or just one. This mechanism is used in the Streaming template to support directing a stream of data to different parts of the Stream Register File as it comes from

main memory. This support is necessary for the high performance of the Streaming template as described in Section 2.1.

Finally, IBM Cyclops VLSI designers have helped us estimate the area cost and design complexity of our M3T micro-architecture. With all this support, we have fully evaluated polymorphism.

3.2 Polymorphous Modeling and Synthesis Tools

The modeling and synthesis tools have leveraged a set of tools originally developed for Adaptive Computing Systems created at Vanderbilt University. These tools have been extended to develop design space representation techniques for fine-grain architectural adaptation, efficient synthesis to polymorphous architectures, and synthesis of both morphable middleware and embedded continuous validation.

These techniques have been verified for both the M3T and the Cyclops architectures.

3.3 Polymorphous Compiler

We have built a source-to-source C compiler to generate polymorphous code. The compiler is based on the Stanford SUIF compiler [HAL96] and the University of Illinois FlexRAM compiler [LEE01,KAN99]. The compiler takes code instrumented with pragmas that indicate how to break the code into modules, what architectural templates to use in each module, how to schedule the modules, and how to overlap execution of the processors. The compiler generates executable code based on these commands.

As shown in Figure 2, the compiler is fully embedded in the M3T polymorphous software tools. In addition, it has two back-ends: one that generates executables for the M3T simulator and one that generates code for the Cyclops simulator. The Cyclops simulator has been enhanced to emulate the polymorphous architecture features of M3T.

3.4 Morphware

The M3T morphware is Linux based, and provides the familiar Posix interface, which allows application programs to be developed (in part) on generic workstations and PCs. The RT-Linux extension from the Embedded Linux Initiative is used and supplemented, thereby allowing precise timing and control of the underlying hardware. However, further optimization and support for architectural adaptivity is necessary.

Morphware configuration is a two-stage process. During system generation, the models of Section 2.2 are used to select and constrain the space of possible API calls. Further, during system generation, the models are used to select and constrain the hardware architecture. Morphware modules are selected only if needed. In this way, the morphware is constrained to a subset of the full API. The model-based configuration selects the minimal set of morphware modules, supporting all hardware and software scenarios for a particular application. In addition, the configuration process synthesizes trigger sources that drive system mode changes (based on the FSM models).

The configuration process builds the morphware system such that the space of possible shapes meets several requirements. The modules chosen at any one instant must be free of conflicts and

must have all of their dependencies met. That is, some modules may not work correctly if other conflicting modules are present. Likewise, other modules may not work correctly if companion modules are not present. The memory footprint must be tracked and maintained. Integration between modules is seamless. After configuration, the M3T compiler (Section 2.3) processes the morphware in order to optimize it, tuning the kernel for the selected hardware template. In this way, each instantiation of the morphware will be a specialized runtime kernel, synthesized directly for the target hardware and application of concern, approaching the efficiency of a hand-rolled kernel. During run time, portions of the morphware move around such that the changing lower- and higher-level interfaces required are supplied as and when needed.

The M3T morphware has been targeted to both the M3T simulator and the Cyclops simulator. The RT-Linux code base provides a good starting point for the Posix interface modules needed to implement the morphware.

3.5 Applications

The application focus for the M3T architecture has been on speech processing applications: one commercial application (Sphinx) and one defense application (LID).

3.5.1 Sphinx

Sphinx [RAV96] is a state-of-the-art popular speech recognition program. It is DARPA funded, and can be downloaded from http://www.speech.cs.cmu.edu/sphinx. It contains 140,000 lines of code. In our experiments, we run it with 9.5 seconds of speech with the Diplomat dictionary, which has 3,493 words. The program takes 12.5 seconds to execute on one processor of a 270 MHz Sgi origin 200.

A Sphinx execution can be divided into the following steps in sequence: (1) signal processing front end, (2) feature distance computation, (3) distance scoring, (4) language model searching, and (5) backtrace.

3.5.2 LID

The LID application is a military application for voice processing. It was provided by military customers of BAE SYSTEMS. LID is used to improve situational awareness and effectiveness through real-time automated aids for the linguist operator. LID automatically sorts signals based on language spoken. It can handle multiple Signals of Interest (SOI) simultaneously.

LID has two key kernels: GMM and Viterbi. GMM implements a Gaussian Mixture Model. It evaluates probabilities formed by a distribution, which is a mixture of Gaussians for all models and all features. It is the most computational part of the LID algorithm. It results in 4 deeply nested loops.

The Viterbi algorithm is used to score paths through the phones for a given feature set. It is not as computationally intensive as the GMM evaluation. It includes a lot of bookkeeping.

4. BACKGROUND AND ALTERNATIVE SOLUTIONS

The M3T system involves not only novel polymorphous micro-architecture, but also integrated and advanced polymorphous software tools. The polymorphous architecture morphs itself into different architectural templates to adapt to different program phases; the software supports polymorphism at each layer of the system: modeling and synthesis tools, compiler, and morphware. To compare with other current technologies, we explore the two aspects separately: architecture and software.

4.1 Architecture

There are several related architectural technologies. We compare our work to (1) chip multiprocessor and intelligent memory technology, (2) reconfigurable computing and (3) hardware support for speculative parallelization.

Chip multiprocessor and intelligent memory

There is significant performance motivation to build wider and larger superscalar chips. However, complexity and cost will soon prevent doing so. CMP (chip multiprocessor) [HAM97] is an advanced technology that puts multiple, relatively simple processors into a single chip. The total area may be comparable to that of a superscalar processor. However, the CMP typically has a higher clock frequency, a simpler implementation, and a lower cost. Examples of CMPs are the IBM Power4 chip and the HP MPOC [RIC02]. Another current trend is the integration of cores and a lot of memory into a single chip, therefore reducing processor-memory communication overhead. This architecture has been called Intelligent Memory. Some examples of such architectures are IRAM [KOZ97], Shamrock [KOG96], Imagine [RIX98], and FlexRAM [KAN99]. Note, however, that conventional CMP and intelligent memory architectures cannot reconfigure the hardware. M3T is a novel CMP architecture that includes new morphable hardware.

Reconfigurable computing

CMU's PipeRench [GOL00][CAD00] is a programmable data-path that can be used to accelerate numerically-intensive applications. PipeRench is composed of pipeline stages called *strips*. The compiler generates the virtual strips and maps them into the hardware strips at run-time. PipeRench is intended to be used as a coprocessor [SCH02] in boards with memory chips and conventional microprocessors. The PipeRench architecture is well suited for regular applications, but its limited reconfiguration makes it difficult to handle irregular ones. CMU's Phoenix project [BUD02] explores the direct implementation of programs in reconfigurable hardware. RFUs (Reconfigurable Functional Units) are working with general purpose processors to improve the program performance. Compiler issues and interface between CPU and RFU are addressed in [VEN02]. Like PipeRench, RFUs are used as coprocessors and the complexity of the interface between CPU and RFU limits its efficiency. In M3T, instead, we use the reconfigurable hardware as the main computation core.

Hardware support for speculative parallelization

Speculative thread-level parallelization [AKK98][CIN00][GAR03][STE00] tries to extract thread level parallelism from hard-to-analyze applications. In this technique, the program is partitioned into threads that are assigned to different execution units. Threads execute in parallel if no dependence violation is detected. If a dependence violation is detected, the offending threads are squashed or repaired. Some hardware support is usually needed for speculative parallelization [SOH95]. Note, however, that misspeculation may hurt performance significantly. In the M3T TaskScalar template, we enforce the dependences between tasks, rather than optimistically assume that there is no violation ahead. Therefore, we minimize task squash overhead.

4.2 Software

On the software side, we compare our work to compiler-supported speculation, the streaming programming model, modeling and synthesis tools, and morphware.

Compiler-supported speculation

With compiler support, speculative thread-level parallelization can significantly improve system performance. Many compiler techniques can be utilized to analyze applications at a high level, resulting in benefits to run-time prediction and speculation. For example, an automatic task selection scheme is proposed in [VIJ98]. The compiler employs control flow heuristics to avoid control flow mis-speculation and employs data dependence heuristics to reduce communication overhead. [KIM01] presents a compiler algorithm to discover a new program property called memory reference idempotency. Memory accesses with such a property can directly access non-speculative storage. They do not need to be buffered, therefore reducing the possibility of speculative storage overflow. The M3T hardware goes beyond this in that it can enforce control and data dependencies at run-time. Thus, the compiler can aggressively generate tasks and let the hardware solve dependence violations.

Stream Programming Model

Several researchers have proposed the stream programming model [MAT01][KAP01]. The model consists of streams and kernels. Streams include sequences of similar data records. Kernels are small programs that operate on the input streams and produce a set of output streams. Imagine [KAP02] is a hardware implementation of this programming model. M3T supports streaming.

Modeling and synthesis tools

The ADAPTERS program of Honeywell [ADA01] addressed system composition using a dataflow mechanism, without provisions for design alternatives (design space is limited to processor assignment). Systems are constructed by assembling components of C and VHDL. Synthesis is accomplished by generating assemblies of components with Genetic algorithms for process placement. BYU synthesizes systems via a new language called Java Hardware Decription Language (JHDL), which allows low-level circuit architecture description [BEL98].

This could form the basis for hierarchical composition of architectures. However, it lacks fine-grained dynamic reconfiguration capabilities. CMU's PipeRench architecture and tools have developed an architecture and compiler technology to map algorithms to a virtual computational pipeline [GOL00]. However, the architecture is restricted to pipeline computation with a fixed ALU component.

The Ptolemy project [EDW97] has investigated multiple 'Models of Computation' and has implemented a design capture for multiple target domains. The system is being extended for synthesis of FPGA-based systems. The Ptolemy project has forged many new concepts in this field, with a variety of representation and analysis formalisms. The models of computation are implemented in a software-intensive process requiring significant effort to tune the modeling paradigm. From the modeling standpoint, these approaches raise the level of abstraction away from the chip-level details. However, they fail to extend to system-wide issues, such as timing, size, weight, and power. They rely on the engineer to translate the model into viable designs for a specific architecture.

The CODEF tool allows design space exploration based on a complete specification of partitioning/scheduling and interconnection synthesis [AUG01]. The focus is on time and area constraints, and primarily targets the design of dedicated systems. The DeFacto project from ISI synthesizes Adaptive Computing Systems (ACS) from source code, using compiler technology, addressing architectural and reconfiguration issues, and targeting fine-grained FPGA architectures [HAL99]. However, these systems fail to extend to system-wide issues, such as timing, size, weight, and power.

M3T is, by definition, not a single architecture but a flexible range of architectures. It needs top-level information and architecture-independent specification in order to be fully utilized. This is what makes our system unique.

Morphware

Morphware concepts have their roots in dynamic reconfiguration of operating systems. Shared libraries and dynamically loaded device drivers of Unix-like operating systems provide concepts in this region [TAN92]. Further, static configuration of operating systems is both an active research [VMW00] and previously successful field. Commercially available real-time operating systems have advanced to a level where intelligent agents (or wizards) aid in the configuration process when targeting a specific embedded platform [WIN99]. Likewise, software optimization of the machine instructions executed at runtime is currently advancing rapidly. For example, Transmeta's Crusoe processor [KLA00] optimizes the VLIW-like instructions it uses to implement the x86 instruction set on the fly through a software reconfiguration.

These related research efforts provide insight into the Morphware challenges, but they do not focus on the same problem. Dynamic reconfiguration of operating systems is focused on the addition of services, but does not consider the shape of the processor hardware changing as it does in a M3T. Static operating system configuration focuses on issues including the target processor hardware architecture, but does not consider a dynamically changing hardware environment. Rather, it expects to use truly hand-coded core routines for every processor architecture. Lower-level concepts modifying the machine code do not expect the operating system shape to morph. For example, Transmeta's layered architecture has all code morphing

well-hidden inside the lowest layers of its architecture, and simply supports existing operating systems at the higher levels.

5. RESULTS, ACCOMPLISHMENTS AND DISCUSSION

In this section, the main results obtained in the M3T project are discussed. The results are organized into the following topics: polymorphous micro-architecture (Section 5.1), IBM Cyclops chip hardware and infrastructure (Section 5.2), performance evaluation (Section 5.3), polymorphous modeling and synthesis tools (Section 5.4), polymorphous compiler (Section 5.5), morphware (Section 5.6), tool integration (Section 5.7), and applications (Section 5.8). For a discussion on technology transfer, see Section 6.

Most of these results are documented in papers that can be obtained from http://iacoma.cs.uiuc.edu/m3t, and the slides presented in PI Meetings and Review Meetings. Consequently, a short summary of the main points is presented here.

5.1 Polymorphous Micro-Architecture

Conceived and designed a set of new micro-architectural primitives for polymorphism that allow a processing architecture such as M3T to dynamically morph into the MIMD, VLIW, TaskScalar, and Streaming architectural templates. As described in Section 2.1, these primitives include the per-cluster Pending Task Window (PTW), the per-processor Task State Table (TST), and the low-overhead synchronization hardware that enables very low overhead multi-group synchronization. As a result, we have an architecture that supports fast core-to-core synchronization, fast core-to-core data communication through registers, and fast core-to-core control communication.

The resulting micro-architecture can morph within and across applications dynamically. Simulation results show that the architecture can deliver at least one order of magnitude higher performance when the architectural template matches the application needs.

Influenced the design of the IBM Blue Gene/C (Cyclops) chip [CAS02] to include several of our micro-architectural supports for polymorphism. Such supports include: (i) very fast barrier synchronization (wired-OR), (ii) support for synchronization groups, (iii) fast access to other thread's local memory, and (iv) interest caching. The last two primitives are very simplified implementations of our PTW and TST primitives.

Developed the novel TaskScalar template which delivers very high performance for irregular codes, and enhances programmer productivity. Programmer productivity is enhanced in three ways: (1) TaskScalar provides hardware support to buffer and deterministically replay code sections, therefore helping debug software bugs [PRV03]; (2) TaskScalar can support atomic code sections, where critical sections are optimistically executed by several threads concurrently. Threads successfully complete if there is no data collision. This support enables TaskScalar to take programs with coarse-grained synchronization and deliver performance as if synchronization had been finely tuned [MAR02]; (3) TaskScalar provides hardware support to detect and fix parallelization bugs, therefore helping the programmer write parallel codes [CEZ03].

Impacted the design of the IBM Productive, Easy-to-use, Reliable Computing System (PERCS) machine concept (part of IPTO's High Productivity Computing Systems program). The proposed PERCS architecture includes some of the TaskScalar supports to enhance programmer productivity, including architectural support for debugging, support for atomic sections, and support to ease parallelization.

Defined the hardware, library, and morphware support necessary to support Streaming in M3T. We defined the architectural support required to support Streaming in M3T. The mapping of the Streaming Virtual Machine (SVM) to an M3T chip is as follows: (1) the Stream Execution Core is the set of CPUs; (2) the Kernel Program Memory is the instruction caches of the CPUs; (3) the Stream Register File is the shared multi-banked cache; (4) the DMA Engine is a plain CPU; and (5) the Control Thread is a plain CPU.

The workings of the SVM are as follows. The Stream Execution Core loads stream inputs from (and stores them to) the shared multi-banked cache, which acts as the Stream Register File (SRF). The Kernel Program Memory works trivially, as it is the instruction caches of the processors in the M3T chip. The SRF allows read and write accesses from both the Stream Execution Core and the Direct Memory Access (DMA) engine, simultaneously. The DMA engine is just another CPU that generates addresses by strides, or indexed if the addresses are recorded in the SRF. It should be noted that the M3T's DMA engine can perform very complex operations, since it is implemented in the M3T architecture with a standard CPU. Finally, the Control Thread gives commands to the Stream Execution Core and the DMA Engine through the M3T's PTW. The Control Thread is notified of task completions (the VM_DONESYNC calls) in two ways: (1) it can build a tree of PTW requests, which allows pipelined streaming, or (2) through the use of barriers.

5.2 Cyclops Chip Hardware and Infrastructure

Completed the design of the soft-core for the Blue Gene/C (Cyclops) chip. The chip includes up to 160 processors with 64-bit wide data-paths [CAS02]. The chip includes some M3T polymorphous hardware as well as additional DOD hardware requirements. The chip will be built in IBM's Cu-11 (0.13 micrometer) ASIC technology (CMOS8S).

Generated the layout of several components of the Cyclops chip. This includes the Floating-Point Unit (FPU) and related logic. The FPU is completely functional. DOD and IBM have signed a 3-year contract to build the hardware and software of a working prototype with Cyclops chips, which will speed up the layout generation.

Developed an initial version of the 64-bit software development tools for the Cyclops chip. These include an assembler, a linker, and a single-threaded instruction set simulator. The simulator is cycle-accurate. The development of C and FORTRAN compilers are currently under way.

Completed the performance characterization of the Cyclops chip architecture. Using the cycle-accurate simulator, detailed performance measurements of key benchmarks were performed on the simulated Cyclops chip. Results demonstrated that multithreading is an effective approach to

tolerate memory latency. Results also demonstrated that a Cyclops chip can achieve 100% peak performance for dense linear algebra kernels and 40% for sparse linear algebra kernels [CAS02].

Started development of multi-chip Cyclops system software. Since the Cyclops chip is expected to be part of a multi-chip machine, the software development effort for such a system has started. A multi-chip debugger environment and the first version of a multi-chip communication library have been developed. The simulator has been upgraded and tested for multi-chip operation. Correctness of first multi-chip application (a molecular dynamics code) has been verified.

5.3 Performance Evaluation

Developed a cycle-by-cycle software simulator of the M3T chip. Major effort was invested in creating a very accurate simulator of the M3T chip with all its templates. This simulator has been used to perform all the evaluations for the project. In addition, the simulator is tied to all the M3T polymorphous software tools. Consequently, it exercises the whole M3T hardware and software system.

Demonstrated significant performance gains by exploiting polymorphism in the Sphinx speech-processing application. The Sphinx application contains three main kernels that can be used to demonstrate the M3T polymorphic capabilities. These kernels and their contribution to the execution time of Sphinx are: GetScores (32%), ChanEval (16%), and CepDist (12%). Each of these kernels runs best using a different template in M3T. Specifically, GetScores is a very regular subroutine and a good match for the VLIW template; CepDist is a parallel kernel and a good match for MIMD; finally, ChanEval is an irregular, loop-less kernel and a good match for TaskScalar. These kernels were executed using the best-matching template on a 4-core M3T, where each core was a 2-issue processor. Without any tuning, a combined speedup of 2.5x relative to Sphinx running on a single M3T core was obtained. This result demonstrates the power of M3T's polymorphism.

Demonstrated the ability of the TaskScalar morph and its compiler pass to speed-up irregular applications beyond the current state-of-the-art. It is well known that the SpecInt applications are highly irregular and difficult to speed-up with current superscalar processors [MAT02]. Consequently, the complete SpecInt2000 applications were executed on the M3T simulator driven by the TaskScalar compiler. The TaskScalar template demonstrated that by using only 2 2-issue cores on the chip and no tuning, a speedup of up to 1.8 over a single superscalar processor equal to one of the cores for the whole application could be obtained.

Demonstrated that Streaming is supported efficiently in M3T: A StreamC FIR kernel running on M3T and on Imagine take comparable numbers of cycles. Streaming support was integrated into the M3T simulator, including the necessary libraries and morphware, demonstrating that Streaming can be supported efficiently in M3T: A StreamC FIR kernel running on M3T and on Imagine took a comparable number of cycles. This was demonstrated at the December 2002 M3T review meeting held in Washington, DC.

Several interesting lessons were learned from the experiment: (1) Few source code modifications are required to support the Imagine code in M3T (The original code's roughly 175 lines are left largely unmodified, and we add about 30 extra lines); (2) The performance achieved on this

particular application required less than a factor of 2 more cycles in the main loops, concluding that M3T can support streaming efficiently; (3) The successful use of StreamC/KernelC was harder than expected; (4) Performance hits can be dramatic if one is not careful in the translation of the inner loops; and (5) Performance debugging is needed throughout the experiments, and with the final PCA products.

Analyzed likely SWEPT improvements of the M3T/Cyclops solution. The SWEPT properties of the current low cost commercial solution for speech processing (Sun Fire 280R server) were compared to a Cyclops chip with 64-bit datapaths. The Sun Fire 280R characteristics are: computation 1800 MIPS, power 600 W, size 1.9 cubic ft, weight 75 lb, can execute 1 real-time channel, and costs about $20,000. For Cyclops, we estimate: computation 64000 MIPS, power 300 W, size 1 cubic ft, weight 15 lb, and can execute 32 real-time channels. Estimating the cost of a Cyclops board that connects to the PCI bus of a PC is hard. However, based on discussions with IBM developers, the cost is estimated to be about $32,000. Consequently, it is estimated that Cyclops provides a ***60x reduction in size, weight, and power per channel***. It is also estimated that Cyclops provides a ***20x reduction in cost per speech channel***. This cost reduction results from the higher integration of the chip and the fact that it exploits parallelism to process several speech channels at the same time.

5.4 Polymorphous Modeling and Synthesis Tools

Developed design space representation tools and techniques for fine-grain architectural adaptation. The dataflow design representation has been extended to support fine-grained execution. Knowledge of the temporal behavior of the component is required. Consequently, the pattern, ordering and timing of data and instruction access is captured. This information is necessary to optimize the location, scheduling, and resource assignment for tasks.

The dataflow representation has been extended to capture significant details of the internal and interface behavior of the components, including:

a) Memory access patterns: footprint, data locality, and critical working sets. These models incorporate a state machine description of a component's primary operating phases along with memory access descriptors for each mode. From this, we are able to determine architectural template suitability.

b) I/O protocol: the signaling protocol, data formats, and timing for data generation. State machine representations are used, with types and sizes of I/O accesses. From these models, we are able to synthesize communications machines (Glue logic).

c) Resource use: registers, CPU functional units, and data-paths. From this, guidance for architectural template selection is derived.

The modeling paradigm and tools capture architectural templates, physical device realizations, and the mapping of logical architectural features to physical structures. The M3T and Cyclops architectures have been modeled. Applications are also modeled. For example, the Sphinx and LID applications described in Section 3.5 were modeled. Finally, the set of morphware components are captured to represent the facilities available to components, along with

performance, resource requirements, and per-template availability. Each hardware and software component permits specifications of constraints for required OS features. Overall, software component models are heavily annotated. The hardware modeling captures substantial details of components and their mapping into architectural templates.

Developed design space exploration tools and techniques for fine-grain architectural adaptation. To explore the design space for system synthesis, Ordered Binary Decision Diagrams (OBDD) [BRY92] were used. To employ OBDDs, the model alternatives are translated into a logical expression, with particular bit fields assigned to represent design alternatives. Each concurrent alternative employs a unique bit field, where exclusive alternatives can share bit fields. Implementation properties, such as timing, power, space, or weight, are discretized to an acceptable resolution and encoded in binary and assigned a bit field. The resulting equation represents the design space.

System constraints are also encoded, and applied in a user-guided order. Constraint application can be a simple AND operator, for constraints involving simple binary exclusion relationships, or can involve implementing complete arithmetic operator logic to compute sums or min/max over all possible combinations of alternatives. As these constraints are successively applied, the design space is reduced until a suitably small set of viable alternatives is obtained.

New model and constraint representation methods have been designed that consider the expansion of the design space. The goal is to reduce the required number of terms. Further operators are added, in addition to the AND and PLUS operators. In critically constrained cases, it is possible that the size grows exponentially. Consequently, the scalability is managed by identifying the onset of OBDD growth and rearranging the OBDD's.

Developed system synthesis tools and techniques for fine-grain architectural adaptation. Architectural selection, component selection, OS/morphware configuration, and online verification generation have been addressed and described below.

a) Architectural Selection. For each computational component, multiple implementations, each one optimized for a particular architectural template, are available. These define a design space, whose bounds are defined by the application flexibility, the span of potential target micro-architectures, and the assignment of components to physical hardware.

b) Component Selection and Connectivity Generation. To achieve maximal performance, the best form is selected, placed on the appropriate hardware component, and connected to data streams. The compiler is used in the loop of the software/hardware generation. As a (computational alternative, architectural template) pair is generated, the compiler is called to produce an implementation and estimate its performance and specific resource utilization. These results are used to select the best implementations within resource constraints, and to verify that the synthesized and compiled system meets requirements.

c) OS/Morphware Configuration and Communication Generation. The application-specific morphware is composed from low-level components. Based on application requirements and hardware architecture, a specific morphware configuration is selected from the potential design space. The configuration contains the range of input API functions and timelines for their use for maximum overlay/multiplexing of resources like memory, CPU, and communication.

d) Online Verification Generation. While embedding instrumentation into the application can detect fault conditions, excessive instrumentation degrades performance. Consequently, monitoring facilities are synthesized to gather only critical information while minimizing impact on performance. Computational modules are generated and installed that: (i) measure constraint-sensitive parameters; (ii) perform on-line computations on those parameters to ensure constraint satisfaction; and (iii) compress and log deviations from the specifications.

5.5 Polymorphous Compiler

Conceived novel compiler pragmas to insert in the code to specify the architectural template to use and its characteristics. These pragmas are either inserted by the programmer or generated automatically by the synthesis tools. Such pragmas are used by the compiler to generate the correct code. There are three groups of pragmas: (1) Architecture reconfiguration pragmas, which indicate how the architecture should be reconfigured at run time. Reconfiguration can occur at any time in the application; (2) Code breakup pragmas, which indicate how the code should be broken down into modules to be scheduled as a unit. The code in a module should have a uniform behavior and good locality; (3) Module scheduling pragmas, which indicate how, when, and where the module should be scheduled. For example, these pragmas may indicate that a module be executed when a certain condition occurs on a certain architectural template.

Developed a source-to-source C compiler to generate polymorphous code. The compiler is based on the Stanford SUIF compiler [HAL96] and the University of Illinois FlexRAM compiler [LEE01,KAN99]. The input to the compiler is code instrumented with pragmas. As shown in Figure 6, the M3T compiler is organized as a set of *transformers*, one for each architectural template. The MIMD and TaskScalar transformers are fully automated. The VLIW and Streaming transformers need sizable hand-holding, as funding was not available to make the transformations completely automatic.

As shown in Figure 2, the compiler is fully embedded in the M3T polymorphous software tools. In addition, it has two back-ends: it generates executables for both the M3T simulator and the Cyclops simulator.

Developed novel compiler algorithms for the TaskScalar template, which are responsible for much of the speedups obtained for irregular applications. The novel TaskScalar transformer includes algorithms to build tasks. To build a task, a four step sequence is followed: (1) high-level transformations, (2) task selection, (3) intra-task optimizations, and (4) inter-task optimizations. The novel inter-task optimizations include task vectorization, task fusion, task fission, task partitioning, task motion, task telescoping, and task elimination. With these algorithms, the TaskScalar transformer is very effective: the complete SpecInt2000 applications were executed on the M3T simulator, and the results showed that the TaskScalar template with 2 cores on the chip and no tuning delivers a speedup of up to 1.8 for the whole application over a single superscalar processor.

5.6 Morphware

Identified and generated about 100 morphware fragments for speech processing. These different fragments have been documented and presented in several PI and Review meetings.

The TAU/Racy tools [MOH94,SHE98] have been ported to the M3T and Cyclops simulators and used for performance measurements. These tools provide a showcase for the power of morphware in speech processing applications.

Actively participated in the Morphware Forum in support of standardization. The M3T team has participated and contributed in a variety of ways to the Morphware Forum. The most significant contribution was in helping to define the software metadata. The M3T design tools have been proposed as a metadata capture system for the Morphware Forum. Their inherent ability to read/write XML, along with other XML/UML authoring tools allows storage, generation, and validation of metadata. Early versions of the tools are available to other PCA teams. Specifically, versions have been given to SPAWAR, Georgia Tech, MPI Software Technology Inc. (MSTI), and MIT. Additionally, Raytheon has expressed interest in using the M3T Model Integrated Computing (MIC) PCA tools for missile system architecture design and validation.

5.7 Tool Integration

Integrated most of the software tool chain. An end-to-end tool flow for some of the architectural templates has been completed. The following sets of tools are integrated:

a) Modeling and synthesis tools, which include component and architectural models (architectural templates and architectural-specific components and metadata) and full-system design space navigation tools (tradeoff size-speed in resources, selection of templates and components, and selection and allocation of resources).

b) Compiler and back-end builder, which generate single-application code that: (1) morphs in time between MIMD, VLIW, Stream, and TaskScalar; (2) space-shares the chip between different templates; and (3) integrates morphware. The code generation tool is also integrated.

c) Simulators for the M3T chip and the Cyclops chip.

The end-to-end path that drives the M3T chip simulator supports the MIMD, Serial, TaskScalar, and (with hand-holding) Streaming templates. The end-to-end path that drives the Cyclops chip simulator supports the MIMD, Serial, and (with hand-holding) VLIW and Streaming templates. Funding cuts prevented the completion of this portion of the effort.

Began supporting code generation for other PCA projects. Support was added to capture state-based system behavior design. Moreover, the OBDD constraints system is being reworked to support XML design space and constraint specification. These tools are useful for other PCA projects. With these tools, parts of the RAW [WAI97] and Smart Memory [MAI00] architectures were modeled. Figure 7 shows how the potential integration of the M3T tools with other PCA projects could be accomplished.

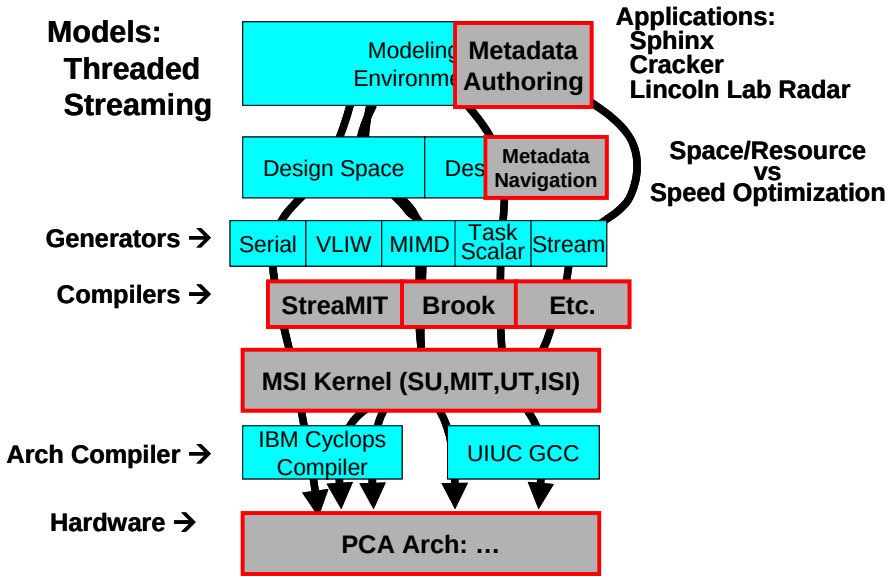


Figure 7: Potential integration of the M3T tools with other PCA projects.

5.8 Applications

Analyzed the Sphinx speech processing application and showed its ability to exploit polymorphism. The Sphinx application was fully characterized (Section 3.5.1). A Sphinx execution has several steps in sequence: (1) signal processing front end, (2) feature distance computation, (3) distance scoring, (4) language model searching, and (5) backtrace. The application was profiled to identify the most time-consuming code sections. These code sections are: GetScores from the distance scoring stage (32% of the Sphinx time), ChanEval from the language model searching stage (16% of the Sphinx time), and CepDist from the feature distance computation stage (12% of the Sphinx time).

Very interestingly, each of these routines has a very different type of code. Consequently, each of these routines can exploit a different template from M3T. Specifically, GetScores is a very regular subroutine and a perfect candidate for the VLIW template; CepDist is a parallel routine and a good match for MIMD; finally, ChanEval is an irregular, loop-less routine that matches the TaskScalar template.

These routines were executed using their best-matching template on a simulated 4-core M3T, where each core was a 2-issue processor. Without any tuning, a combined speedup of 2.5x relative to Sphinx running on a single, simulated M3T core was obtained. This result demonstrates the power of M3T's polymorphism for speech processing applications.

Analyzed the LID voice processing military application and showed its ability to exploit polymorphism. The LID application was also fully analyzed (Section 3.5.2) and its execution

was profiled. Its two main kernels are GMM and Viterbi. GMM implements a Gaussian Mixture Model. It evaluates probabilities formed by a distribution, which is a mixture of Gaussians for all models and all features. The Viterbi algorithm is used to score paths through the phones for a given feature set. It is not as computationally intensive as the GMM evaluation.

Looking at the code, GMM corresponds to the combination of GetScores and CepDist in Sphinx. Viterbi corresponds to ChanEval in Sphinx. Consequently, once again, different M3T templates work best on different subroutines: GMM runs best under the MIMD and VLIW templates, while Viterbi runs best under the TaskScalar template. This proves again the power of M3T for speech processing applications. Unfortunately, it was not possible to quantify the speedups, if any, for the LID application due to programmatic funding limitations beyond our control.

Estimated likely SWEPT improvements of the M3T/Cyclops solution for speech processing applications. The SWEPT properties of the current low cost commercial solution for speech processing (Sun Fire 280R server) were compared to a Cyclops chip with 64-bit datapaths. The Sun Fire 280R characteristics are: computation 1800 MIPS, power 600 W, size 1.9 cubic ft, weight 75 lb, can execute 1 real-time channel, and costs about $20,000. For Cyclops, we estimate: computation 64000 MIPS, power 300 W, size 1 cubic ft, weight 15 lb, can execute 32 real-time channels, and costs about $32,000. Consequently, it is estimated that Cyclops provides a 60x reduction in size, weight, and power per channel. Additionally Cyclops provides an estimated 20x reduction in cost per channel.

6. TECHNOLOGY TRANSFER

The M3T project has developed technology that has been transferred to industry. In this section, the main technology transfer accomplishments are summarized.

Impacted the architecture of the IBM Blue Gene/C (Cyclops) chip. IBM's Cyclops architecture now includes several polymorphous features originally proposed for M3T, including: (i) very fast barrier synchronization (wired-OR), (ii) support for synchronization groups, (iii) fast access to other thread's local memory, and (iv) interest caching. These last two primitives are very simplified implementations of our PTW and TST primitives. M3T concepts have influenced IBM computer architects.

M3T TaskScalar's architectural supports to enhance programmer productivity have impacted the IBM PERCS machine concept (part of IPTO's HPCS program). Specifically, the currently-proposed PERCS architecture includes some of TaskScalar's architectural support for debugging [PRV03], support for atomic sections [MAR02], and support to ease parallelization [CEZ03].

Impacted BAE SYSTEMS customer base, which is now aware of the possibilities of polymorphous architectures to speed up speech processing. A solution based on a 64-bit Cyclops/M3T chip is estimated to deliver a 60x reduction in size, weight, and power per speech channel, and a 20x reduction in cost per speech channel, compared to the current low-cost solution for speech processing.

Impacted the Morphware Forum. The M3T design tools are being proposed as a metadata capture system for the PCA program's Morphware Forum. The tools' inherent ability to read/write XML, along with other XML/UML authoring tools allows storage, generation, and validation of metadata. Early versions of the tools are available to other teams in IPTO's PCA program. Specifically, versions have been given to SPAWAR, Georgia Tech, MSTI, and MIT.

Raytheon is considering using Model Integrated Computing (MIC) PCA tools from M3T for a missile system architecture design and validation.

The Model Integrated Computing Alliance at ISIS, which includes Boeing, GM/Saturn, DuPont, Motorola, USAF/AEDC, USARMY/AMCOM and NASA is now aware of the power of PCA architectures in general and the M3T architecture in particular. Personnel in instrumentation and image processing for United States Air Force Arnold Engineering Development Center (USAF/AEDC) have indicated their interest in M3T technology for aerospace testing applications. Current work with AEDC for image processing should benefit from PCA concepts. The M3T architecture is also well suited to low-power wireless applications. Ongoing work at ISIS in the PCES program aimed at system synthesis from a Waveform Definition Language has been interfacing with the Air Force Research Laboratory's Software Radio group at Rome, NY (AFRL/IFG). The M3T architecture is naturally suited to multi-band/multi-mode radio applications. Motorola has also expressed interest in the commercial applications of this technology for cell phones and digital radio.

7. CONCLUSIONS AND RECOMMENDATIONS

The Morphable Multithreaded Memory Tiles (M3T) system is a novel computer system that has proved that a computer system with both polymorphous hardware and a complete polymorphous software environment is orders of magnitude more cost-effective than conventional computing systems. M3T innovates with two main ideas: (1) novel polymorphous architectural support that enables the M3T chip to appear as a TaskScalar, VLIW, MIMD, or Streaming engine (or a time/space shared combination of them) as the application runs, and (2) a complete polymorphous software environment that includes synthesis tools for high-level design, a polymorphous compiler and code generator, and morphware fragments.

The accomplishments of the M3T project have been many. The contributions in micro-architecture have been the conception and design of a set of new micro-architectural primitives for polymorphism; the development of the novel TaskScalar template which, in addition to delivering very high performance for irregular codes, enhances programmer productivity; the definition of the hardware, library, and morphware support for Streaming in M3T; the impact on the design of the IBM Blue Gene/C (Cyclops) chip, which includes several of M3T's micro-architectural supports for polymorphism; and the impact on the design of the IBM PERCS machine concept (part of IPTO's HPCS program).

The contributions in the Cyclops chip hardware and infrastructure have been the completion of the design of the soft-core for the Blue Gene/C (Cyclops) chip; the generation of the layout of several components of the Cyclops chip; the development of an initial version of the 64-bit software development tools for the Cyclops chip; the performance characterization of the Cyclops chip architecture, and the partial development of multi-chip Cyclops system software.

As for performance evaluation, a cycle-by-cycle software simulator of the M3T chip has been developed; the ability of the TaskScalar template and its compiler pass to speed-up irregular applications beyond the current state-of-the-art has been demonstrated; Streaming is now supported efficiently in M3T; and significant performance gains in the Sphinx speech-processing application have been shown by exploiting the M3T's polymorphic capabilities.

The contributions in polymorphous modeling and synthesis tools have been the development of design space representation tools for fine-grain architectural adaptation; and the development of novel design space exploration and system synthesis techniques for architectural adaptation. As for the polymorphous compiler, novel pragmas that specify the architectural template and its characteristics have been conceived; a source-to-source C compiler that generates polymorphous code has been developed; and novel compiler algorithms for the TaskScalar template that are responsible for much of the speedups obtained for irregular applications have been developed. Moreover, about 100 morphware fragments for speech processing have been identified and generated, and active participation in the Morphware Forum has resulted in standardization of concepts. After substantial effort, most of the M3T's software tool chain has been integrated together, and code generation for other PCA projects has started.

Finally, the contributions in applications have been the analysis of the Sphinx and LID speech processing applications. The ability of these applications to exploit polymorphism has been demonstrated; and the likely very large SWEPT improvements of the M3T/Cyclops solution for speech processing applications have been estimated.

An important emphasis of the M3T project has been technology transfer. The architecture of the IBM Blue Gene/C (Cyclops) chip has been influenced by the insertion of polymorphous micro-architecture constructs; the design of the IBM PERCS machine concept (part of the HPCS program) has been impacted with TaskScalar's architectural supports to enhance programmer productivity; BAE SYSTEMS customer base, is now aware of the benefits that polymorphous architectures can provide to speed up speech processing; the Morphware Forum has also been influenced; a project with Raytheon to use MIC PCA tools for missile system architecture design and validation has been developed; and the companies in the Model Integrated Computing Alliance at ISIS have been impacted.

It is strongly recommend that further study of **M3T's TaskScalar template** is performed. This is a novel template that enhances **programmer productivity**, in addition to delivering very high performance for irregular codes. Programmer productivity is enhanced in several ways. For example, TaskScalar provides hardware support to buffer and deterministically replay code sections, which greatly helps debugging software bugs [PRV03]. In addition, TaskScalar supports atomic code sections, where critical sections are optimistically executed by several threads concurrently. These sections successfully complete if there is no data collision. This support enables TaskScalar to take programs with coarse-grained synchronization and deliver performance as if synchronization had been finely tuned [MAR02]. Finally, TaskScalar provides hardware support to detect and fix parallelization bugs, therefore assisting the programmer as parallel codes are developed [CEZ03].

It is also recommended that **the tool integration** provided by the M3T project be exploited in as many ways as possible. The set of polymorphous software tools put together in the M3T project **is unique**. It includes modeling and synthesis tools, compiler, back-end builder, and morphware. These tools have been proven to boost the capabilities of polymorphous architectures. They should be used in other PCA architectures.

REFERENCES

1. [ADA01] http://www.htc.honeywell.com/projects/adapters/. (2001) "The Honeywell ADAPTERS Project", June.

2. [AKK98] Akkary, H. and Driscoll, M. A. (1998) "A Dynamic Multithreading Processor," *International Symposium on Microarchitecture*, December.

3. [AUG01] Auguin M., Capella L., Cuesta, F., and Gresset, E. (2001) "CODEF: A System Level Design Space Exploration Tool," *Intl. Conf. on Acoustics, Speech, and Signal Processing*.

4. [BAP00] Bapty, T., Neema S., Scott J., Sztipanovits J., and Assad, S. (2000) "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," *VLSI Design*, Vol. 10, No. 3, pp. 281-306.

5. [BEL98] Bellows, P., and Hutchings, B. (1998) "JHDL – An HDL for Reconfigurable Systems," *Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines*, pp. 175-184, April.

6. [BRY92] Bryant, R.E. (1992) "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," Technical Report CMU-CS-92-160, School of Computer Science, Carnegie Mellon University, June.

7. [BUD02] Budiu, M., Mishra, M., Bharambe, A., and Goldstein, S. C. (2002) "Peer-to-peer Hardware-software Interfaces for Reconfigurable Fabrics," *IEEE Symposium on Field-Programmable Custom Computing Machines*, April.

8. [CAD00] Cadambi S., and Goldstein, S.C. (2000) "Fast and Efficient Place and Route for Pipeline Reconfigurable Architectures," *Proceedings of IEEE International Conference on CAD*.

9. [CAS02] Cascaval, C., Castanos, J., Ceze, L., Denneau, M., Gupta, M., Lieber, D., Moreira, J., Strauss, K., and Warren, H. (2002) "Evaluation of a Multithreaded Architecture for Cellular Computing," *Eighth International Symposium on High-Performance Computer Architecture (HPCA), February*.

10. [CEZ03] Ceze, L., Strauss, K., Teodorescu, R., and Torrellas, J. (2003) "Program Parallelization Assisted by Thread-Level Speculation," University of Illinois Technical Reports, May.

11. [CIN00] Cintra, M., Martinez, J. F., and Torrellas, J. (2000) "Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors," *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June.

12. [EDW97] Edwards, S., Lavagno, L., Lee, E. A., and Sangiovanni-Vincentelli, A. (1997) "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proceedings of the IEEE*, Vol. 85, No. 3, March.

13. [GAR03] Garzaran, M., Prvulovic, M., Llaberia, J., Vinals, V., Rauchwerger, L., and Torrellas, J. (2003) "Tradeoffs in Buffering Memory State for Thread-Level Speculation in

Multiprocessors," *Ninth International Symposium on High-Performance Computer Architecture (HPCA), February*.

14. [GOL00] Goldstein, S. C., Schmit, H., Budiu, M., Cadambi, S., Moe, M., and Taylor, R. R. (2000) "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, Vol.33, No. 4*,* April.

15. [HAL96] Hall, M., Anderson, J., Amarasinghe, S., Murphy, B., Liao, S., Bugnion, E., and Lam, M. (1996) "Maximizing Multiprocessor Performance with the SUIF Compiler," *IEEE Computer, December*.

16. [HAL99] Hall, M., Diniz, P., Bondalapati, K., Ziegler, H., Duncan, P., Jain, R., and Granacki, J. (1999) "DEFACTO: A Design Environment for Adaptive Computing Technology," *Proceedings of the 6th Reconfigurable Architectures Workshop (RAW'99).*

17. [HAM97] Hammond, L., Nayfeh, B. A., and Olukotun, K. (1997) "A Single-chip Multiprocessor," *Computer*, Sep, pp79-85.

18. [HAR] Harel, D. "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming,* 8, pp. 231-278.

19. [HUA00] Huang, M., Renau, J., Yoo, S.-M., and Torrellas, J. (2000) "A Framework for Dynamic Energy Efficiency and Temperature Management," *33rd International Symposium on Microarchitecture (MICRO)*, December.

20. [KAN99] Kang, Y., Huang, M., Yoo, S.-M., Ge, Z., Keen, D., Lam, V., Pattnaik, P., and J. Torrellas, J. (1999) "FlexRAM: An Advanced Intelligent Memory System," *International Conference on Computer Design (ICCD),* October.

21. [KAP01] Kapasi, U. J., Mattson, P., Dally, W. J., Owens, J. D. and Towles, B. (2001) "Stream Scheduling," *Proceedings of the 3rd Workshop on Media and Streaming Processors*, December.

22. [KAP02] Kapasi, U. J., Dally, W. J., Rixner, S., Owens, J. D., and Khailany, B. (2002). "The Imagine Stream Processor," *Proceedings of International Conference on Computer Design*, September.

23. [KIM01] Kim, S. W., Ooi, C., Eigenmann, R., Falsafi, B. and Vijaykumar, T. N. (2001) "Reference Idempotency Analysis: A Framework for Optimizing Speculative Execution," *PPoPP'01*, June.

24. [KLA00] Klaiber, A. (2000) "The Technology Behind Crusoe Processors," Transmeta Corporation White Paper, Santa Clara, CA. (www.transmeta.com)

25. [KOG96] Kogge, P., Bass, S., Brockman, J., Chen, D., and Sha, E. (1996) "Pursuing a Petaflop: Point Designs for 100 TF Computers Using PIM Technologies," *Frontiers of Massively Parallel Computation Symposium*.

26. [KOZ97] Kozyrakis, C., Perissakis, S., Patterson, D., Anderson, T., Asanovic, K., Cardwell, N., Fromm, R., Golbus, J., Gribstad, B., Keeton, K., Thomas, R., Treuhaft, N., and Yelick, K. (1997) "Scalable Processors in the Billion-Transistor Era: IRAM." *IEEE Computer*, September.

27. [LEE01] Lee, J., Solihin, Y., and Torrellas, J. (2001) "Automatically Mapping Code in an Intelligent Memory Architecture," *Seventh International Symposium on High-Performance Computer Architecture (HPCA)*, January.

28. [MAI00] Mai, K., Paaske, T., Jayasena, N., Ho, R., and Horowitz, M. (2000) "Smart Memories: A Modular Reconfigurable Architecture," *International Symposium on Computer Architecture (ISCA)*, June.

29. [MAR02] Martinez, J., and Torrellas, J. (2002) "Speculative Synchronization: Applying Thread-Level Speculation to Parallel Applications," *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), October*.

30. [MAT02] Martinez, J., Renau, J., Huang, M., Prvulovic, M., and Torrellas, J. (2002) "Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors," *35th International Conference on Microarchitecture (MICRO),* November.

31. [MAT01] Mattson, P. (2001) "A Programming System for the Imagine Media Processor," *Ph.D. Thesis, Dept. of Electrical Engineering, Stanford University.*

32. [MOH94] Mohr, B., Brown, and D., Malony, A. (1994) "TAU: A Portable Parallel Program Analysis Environment for pC++," *Proceedings of CONPAR 94,* September.

33. [NAR95] Narayan, S. and Gajski, D. (1995) "Interfacing Incompatible Protocols Using Interface Process Generation," *32nd ACM/IEEE Design Automation Conference*, 1995.

34. [PAS98] Passerone, R., Sangiovanni-Vincentelli, A., and Rowson, J. (1998) "Automatic Synthesis of Interfaces between Incompatible Protocols," *35th Design Automation Conference*.

35. [PRV03] Prvulovic, M., and Torrellas, J. (2003) "ReEnact: Using Thread-Level Speculation to Debug Data Races in Multithreaded Codes," *30th International Symposium on Computer Architecture (ISCA), June*.

36. [RAV96] Ravishankar, M. (1996) "Efficient Algorithms for Speech Recognition," Ph.D. Thesis, Carnegie-Mellon University.

37. [RIC02] Richardson, S. (2002) "MPOC: A Chip Multiprocessor for Embedded Systems," HP Labs Technical Reports, HP-2002-186.

38. [RIX98] Rixner, S., Dally, W.J., Kapasi, U.J., Khailany, B., Lopez-Lagunas, A., Mattson, P.R., and Owens, J.D. (1998) "A Bandwidth-Efficient Architecture for Media Processing," *31st International Symposium on Microarchitecture*, November.

39. [SCH02] Schmit, H., Whelihan, D., Tsai, A., Moe, M., Levine, B., and Taylor, R. R. (2002) "PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology," *IEEE Custom Integrated Circuits Conference (CICC).*

40. [SER99] Service, R. F. (1999) "Big Blue Aims to Crack Protein Folding Riddle," *Science,* pp. 2250, December.

41. [SHE98] Shende, S., Malony, A., Cuny, J., Lindlan, K., Beckman, P., and Karmesin, S. (1998) "Portable Profiling and Tracing for Parallel Scientific Applications using C++," *Proceedings of Symposium on Parallel and Distributed Tools*, August.

42. [SOH95] Sohi, G., Breach, S. and Vajapeyam, S. (1995) "Multiscalar Processors," *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June.

43. [SPE00] http://www.spec.org (2000) "The Standard Performance Evaluation Corporation (SPEC)".

44. [STE00] Steffan, J. G., Colohan, C. B., Zhai, A. and Mowry, T. C. (2000) "A Scalable Approach to Thread-level Speculation," *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June.

45. [SVM03] http://www.morphware.org (2003) "PCA Morphware Virtual Machine Specification", August.

46. [TAN92] Tanenbaum, A. (1992) *Modern Operating Systems*. Prentice Hall.

47. [TRE99] Tremblay, M. (1999) "MAJC: Microprocessor Architecture for Java Computing," presentation at *Hot Chips*, August.

48. [VEN02] Venkataramani, G., Sudhir, S., Budiu, M., and Goldstein, S. C. (2002) "Factors Influencing the Performance of a CPU-RFU Hybrid Architecture," *12th International Conference on Field Programmable Logic and Applications*, September.

49. [VIJ98] Vijaykumar T. N., and Sohi, G. S. (1998) "Task Selection for a Multiscalar Processor," *31st International Symposium on Microarchitecture (MICRO-31)*, November.

50. [VMW00] VMWARE (2000) *Linux Configuration Wizard*. VMWARE Inc., Palo Alto, CA. (www.vmware.com)

51. [WAI97] Waingold, E., Taylor, M., Srikrishna, D., Sarkar, V., Lee, W., Frank, M., Finch, P., Barua, R., Babb, J., Amarasinghe, S., and Agarwal, A. (1997) "Baring it All to Software: Raw Machines." *IEEE Computer*, September.

52. [WIN99] Wind River Corporation (1999) *Tornado User's Guide*. Wind River Corporation, Alameda, CA.

LIST OF SYMBOLS

API: Application Program Interface
DMA: Direct Memory Access
HPCS: High Productivity Computing Systems
LID: Language Identifier
M3T: Morphable Multithreaded Memory Tiles
MIC: Model Integrated Computing
MIMD: Multiple Instruction Streams and Multiple Data Steams
OBDD: Ordered Binary Decision Diagrams
PCA: Polymorphous Computing Architecture
PERCS: Productive, Easy-to-use, Reliable Computing System
PTW: Pending Task Window
SUIF: Stanford University Intermediate Format
SVM: Streaming Virtual Machine
TST: Task State Table
VLIW: Very Long Instruction Word